**Discrete Key Value Bottleneck**

**Code Implementation Review**

**First things first**   
**Link to repository hosting my source code :** <https://github.com/chetxn04/Discrete_KV_Bottleneck.git>

**Code overview :**

The repository has two files. The main focus of this code review is going to be on the file titled [**NUS\_Final.ipynb**](https://github.com/chetxn04/Discrete_KV_Bottleneck/blob/main/NUS_Final.ipynb) (The second file will be discussed towards the end). The code demonstrates the application of a Discrete Key-Value Bottleneck model for continual learning using the well-known MNIST dataset for classification tasks. **The implementation is done from scratch.**

**Main Classes and Functions**:

1. **NonStationaryMNIST**: Manages the non-stationary MNIST dataset for incremental learning by providing access to the data in a class-incremental manner.
2. **SimpleMLP**: Implements a basic multi-layer perceptron model for classification tasks on the MNIST dataset.
3. **DiscreteKeyValueBottleneck**: Defines the architecture and functionality of the discrete key-value bottleneck mechanism for improved memory retention in continual learning.
4. **KeyValueBottleneckModel**: Combines the discrete key-value bottleneck with a neural network for effective classification in a continual learning setting.
5. **Initialize\_keys\_data\_aware**: Initializes key representations in a data-aware manner to enhance the model's ability to retain information over time.
6. **Train\_model\_with\_logging**: Trains the specified model while logging performance metrics for analysis and visualization.

**Results, Analysis & Interpretation :**

Following is a log of the changes in loss of both the MLP and DBKV model.

Device: cpu

Loading MNIST dataset...

**Training with classes: 0 and 1**

[KV Bottleneck] Step [100/1000], Loss: 8.7827

[KV Bottleneck] Step [200/1000], Loss: 0.0542

[KV Bottleneck] Step [300/1000], Loss: 0.0495

[KV Bottleneck] Step [400/1000], Loss: 0.0487

[KV Bottleneck] Step [500/1000], Loss: 0.0484

[KV Bottleneck] Step [600/1000], Loss: 0.0485

[KV Bottleneck] Step [700/1000], Loss: 0.0487

[KV Bottleneck] Step [800/1000], Loss: 0.0482

[KV Bottleneck] Step [900/1000], Loss: 0.0490

[KV Bottleneck] Step [1000/1000], Loss: 0.0480

[MLP] Step [100/1000], Loss: 1.1683

[MLP] Step [200/1000], Loss: 0.0022

[MLP] Step [300/1000], Loss: 0.0001

[MLP] Step [400/1000], Loss: 0.0000

[MLP] Step [500/1000], Loss: 0.0000

[MLP] Step [600/1000], Loss: 0.0851

[MLP] Step [700/1000], Loss: 0.0000

[MLP] Step [800/1000], Loss: 0.0000

[MLP] Step [900/1000], Loss: 0.0000

[MLP] Step [1000/1000], Loss: 0.0000

**Training with classes: 2 and 3**

[KV Bottleneck] Step [100/1000], Loss: 0.0478

[KV Bottleneck] Step [200/1000], Loss: 0.0478

[KV Bottleneck] Step [300/1000], Loss: 0.0477

[KV Bottleneck] Step [400/1000], Loss: 0.0473

[KV Bottleneck] Step [500/1000], Loss: 0.0479

[KV Bottleneck] Step [600/1000], Loss: 0.0472

[KV Bottleneck] Step [700/1000], Loss: 0.0475

[KV Bottleneck] Step [800/1000], Loss: 0.0476

[KV Bottleneck] Step [900/1000], Loss: 0.0470

[KV Bottleneck] Step [1000/1000], Loss: 0.0476

[MLP] Step [100/1000], Loss: 0.0671

[MLP] Step [200/1000], Loss: 0.0000

[MLP] Step [300/1000], Loss: 0.0000

[MLP] Step [400/1000], Loss: 0.0000

[MLP] Step [500/1000], Loss: 0.0000

[MLP] Step [600/1000], Loss: 0.0000

[MLP] Step [700/1000], Loss: 0.1169

[MLP] Step [800/1000], Loss: 0.0000

[MLP] Step [900/1000], Loss: 0.0000

[MLP] Step [1000/1000], Loss: 0.0000

**Training with classes: 4 and 5**

[KV Bottleneck] Step [100/1000], Loss: 0.0474

[KV Bottleneck] Step [200/1000], Loss: 0.0473

[KV Bottleneck] Step [300/1000], Loss: 0.0471

[KV Bottleneck] Step [400/1000], Loss: 0.0471

[KV Bottleneck] Step [500/1000], Loss: 0.0463

[KV Bottleneck] Step [600/1000], Loss: 0.0462

[KV Bottleneck] Step [700/1000], Loss: 0.0467

[KV Bottleneck] Step [800/1000], Loss: 0.0466

[KV Bottleneck] Step [900/1000], Loss: 0.0464

[KV Bottleneck] Step [1000/1000], Loss: 0.0462

[MLP] Step [100/1000], Loss: 0.0000

[MLP] Step [200/1000], Loss: 0.0522

[MLP] Step [300/1000], Loss: 0.0000

[MLP] Step [400/1000], Loss: 0.0000

[MLP] Step [500/1000], Loss: 0.0000

[MLP] Step [600/1000], Loss: 0.0000

[MLP] Step [700/1000], Loss: 0.0000

[MLP] Step [800/1000], Loss: 0.1227

[MLP] Step [900/1000], Loss: 0.0000

[MLP] Step [1000/1000], Loss: 0.0000

**Training with classes: 6 and 7**

[KV Bottleneck] Step [100/1000], Loss: 0.0457

[KV Bottleneck] Step [200/1000], Loss: 0.0463

[KV Bottleneck] Step [300/1000], Loss: 0.0458

[KV Bottleneck] Step [400/1000], Loss: 0.0457

[KV Bottleneck] Step [500/1000], Loss: 0.0455

[KV Bottleneck] Step [600/1000], Loss: 0.0455

[KV Bottleneck] Step [700/1000], Loss: 0.0453

[KV Bottleneck] Step [800/1000], Loss: 0.0452

[KV Bottleneck] Step [900/1000], Loss: 0.0454

[KV Bottleneck] Step [1000/1000], Loss: 0.0454

[MLP] Step [100/1000], Loss: 0.0000

[MLP] Step [200/1000], Loss: 0.0000

[MLP] Step [300/1000], Loss: 0.0000

[MLP] Step [400/1000], Loss: 0.0655

[MLP] Step [500/1000], Loss: 0.0000

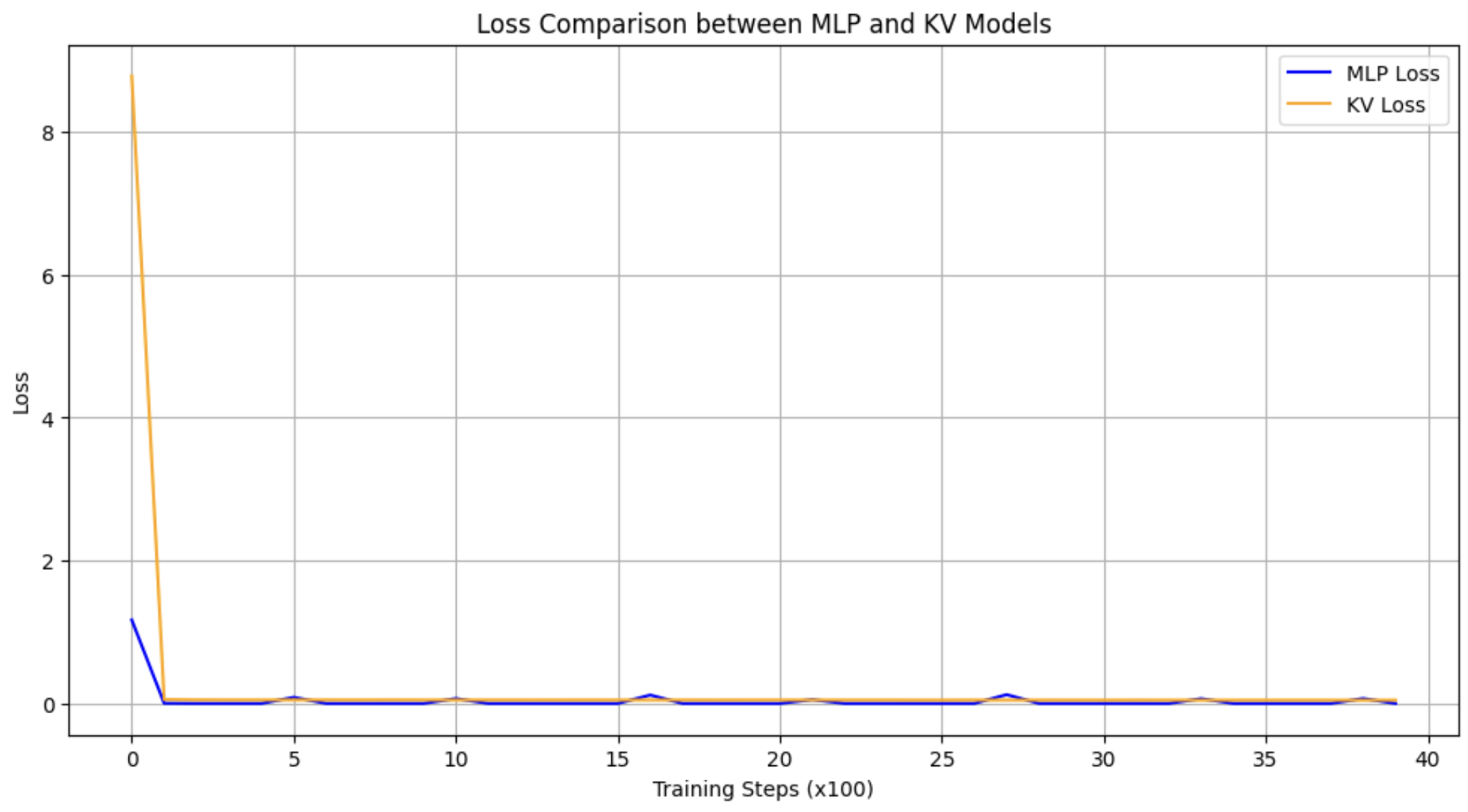
[MLP] Step [600/1000], Loss: 0.0000

[MLP] Step [700/1000], Loss: 0.0000

[MLP] Step [800/1000], Loss: 0.0000

[MLP] Step [900/1000], Loss: 0.0678

[MLP] Step [1000/1000], Loss: 0.0000



**My inferences :-**

* The **KV model** shows relatively stable loss values throughout the training process, with small fluctuations. It starts with a higher loss at the beginning of each task but quickly settles down to lower values (around 0.045 - 0.048), suggesting it is effectively learning the new tasks.I suppose the higher initial loss could be lowered by better initialization
* The consistency in the loss values across different tasks suggests that the KV bottleneck may be more robust in retaining previously learned knowledge while still being able to learn new tasks. It could also be a sign of plateauing.
* There are some spikes in the loss for the MLP model during specific steps. These spikes could indicate that the MLP is struggling to retain knowledge when transitioning to new tasks, reflecting catastrophic forgetting. (Just my guess trying to find a reason for the unexpected upticks in loss)
* The MLP model demonstrates drastic changes in its loss values across training steps. It often reaches a near-zero loss very quickly, indicating that it might be overfitting to the current task.

**A few very important points that need to be mentioned.**

The current specification of the bottleneck has shown the best results till now. I have spent a lot of time trying to figure out what might be better. Experimented with :

1. **Number of keys in the bottleneck** (Tried from **10** till current **2000**). I find it fairly safe to say that an increase in the number of keys has led to lower loss. Hence better results
2. **Learning rate** (Tried from **0.01** to **0.001** on various values). Have landed on 0.0025 to be the ideal. Needs more work , some other value might work better. Could work on a **variable learning rate.**
3. **Number of examples of each class** that were presented to the model (**100** to **6,000** which is basically the e**ntire MNIST training dataset** for a class). Exposure to more variations increases the performance of the model.
4. **Number of steps for which these examples** were presented to the model **(1000 to 2000).** A very computationally expensive operation.
5. **Variations of activation functions.** Started from **random initialization** , random **gaussian projection matrices** based initialization , **K means clustering** based on input feature vectors, feature selectors (visited this only in theory) , and then data aware initialization finally . **Data-Aware Initialization** is a strategy that leverages information from a related dataset to initialize the model's parameters (keys in the KV bottleneck), allowing the model to start with a more informed representation of the input space. This approach works because it helps the model quickly adapt to new tasks by providing a foundational understanding of the data distribution. For the MNIST dataset I used sample images from each class as the keys (after feature reduction to fit the bottleneck). Credit to GPT for suggesting this. **In my opinion, initialization can be improved the most upon, to reduce the initially high losses.**
6. **Dimension of the keys.** (all the way from 2 to currently 32). Increase in dimension of keys leads to better representation and more precise matching in my opinion.

The following were the final values of the validation loss of both the models   
1) MLP Validation Loss: **64.7455**

2) KV Validation Loss: **10.0810**

This indicates that the KV model is better at capturing the patterns in the validation dataset. It suggests that the KV model suffers many times less catastrophic forgetting than MLP.   
  
However the loss of 10.08 in my opinion is quite high and needs more work to decrease it. This requires more hours put into figuring out the ideal configuration of above mentioned parameters.

A short note on the next file titled [**NUS\_Bigger\_Experiment\_2.ipynb**](https://github.com/chetxn04/Discrete_KV_Bottleneck/blob/main/NUS_Bigger_Experiment_2.ipynb)**.**

This code was my attempt on recreating the experiment described in section 5.2 of the paper. I have written the code for **initialization of keys** (using CIFAR 100 and ResNET to extract useful features) and **EMA** to correct the keys during initialization, creating **projection vectors** using **random gaussian projection matrices**, importing the encoder , decoder and fitting all of these together to create the **complete model**.

However the project frankly required a lot of resources as the specifications described in the paper were **256 codebooks with 4096 key-value pairs each, 14-dimensional keys and 10-dimensional values.** Training this model with ResNET 50 as an encoder for 2000 epochs for each set is in my opinion was beyond the time & resources available to me (Google Colab runtime disconnection , Limited GPU units etc ..). I hope it's understandable.

**Citations :**

1. [arXiv:2207.11240v3](https://arxiv.org/abs/2207.11240v3)
2. Google
3. ChatGPT

**Declaration** :

All the codes and the reproduction of experiments have been done on my own based on my understanding of the paper. My only sources are the paper and information provided by Google and ChatGPT.

**Chetan Maheshwari**

**4th Year Undergraduate**

**Department of Electronics and Communication**

**National Institute of Technology , Karnataka**

**Pin Code : 575025**

**India**